Time-predictable Chip-Multiprocessor Design

Schoeberl, Martin

Published in:
Asilomar Conference on Signals, Systems, and Computers

Publication date:
2010

Document Version
Early version, also known as pre-print

Time-predictable Chip-Multiprocessor Design

Martin Schoeberl
Department of Informatics and Mathematical Modeling
Technical University of Denmark
Email: masca@imm.dtu.dk

***Abstract*—Real-time systems need time-predictable platforms to enable static worst-case execution time (WCET) analysis. Improving the processor performance with superscalar techniques makes static WCET analysis practically impossible. However, most real-time systems are multi-threaded applications and performance can be improved by using several processor cores on a single chip. In this paper we present a time-predictable chip-multiprocessor system that aims to improve system performance while still enabling WCET analysis.**

The proposed chip-multiprocessor (CMP) uses a shared memory with a time-division multiple access (TDMA) based memory access scheduling. The static TDMA schedule can be integrated into the WCET analysis. Experiments with a JOP based CMP showed that the memory access starts to dominate the execution time when using more than 4 processor cores. To provide a better scalability, more local memories have to be used. We add a processor local scratchpad memory and split data caches, which are still time-predictable, to the processor cores.

I. Introduction

Research on time-predictable processor architectures is gaining momentum. The research is driven by the issues of worst-case execution time (WCET) analysis for current processors. Standard processors are optimized for average case performance and many of the average case performance enhancing features are hard to integrate into static WCET analysis. The most important features, such as caches and dynamic branch predictors, contain a lot of state information that depends on the execution history. While exactly this state increases the performance, it is not feasible to track the concrete state in static program analysis. The state needs to be abstracted and due to the information loss in the abstract state, the analysis has to make conservative assumptions, such as predicting a cache miss when the cache content is not known. However, with out-of-order processors even this assumption is not safe. Due to timing anomalies [12] this local worst case does not need to trigger the global WCET. It has been shown that a cache hit can actually lead to a higher execution time than a cache miss. Those architectures are not timing compositional and lead to a state space explosion for static WCET analysis.

A time-predictable processor is designed to enable and simplify WCET analysis [21]. Only analyzable performance enhancing features shall be implemented. For example, the pipeline and cache have to be organized to be timing compositional to enable independent pipeline and cache analysis. The optimization of the processor is on the WCET instead of average case performance. For hard real-time systems the average case performance does not matter at all, only the WCET is important.

As a result such a processor will be slower in the average case than a *standard* processor. To reconcile performance with predictability we argue to build chip-multiprocessor (CMP) systems. As some of the resource hungry features, which are hard to analyze, are dropped from the processor design, the transistors are better spent by replicating simple pipelines on a single chip. The remaining issue is to build a CMP system where the access to the shared resource main memory can be performed time-predictable. The execution time of the threads running on the different cores shall not depend on each other. Therefore, the access to main memory has to be scheduled statically with a time-division multiple access (TDMA) based arbitration scheme.

Embedded applications need to control and interact with the *real* world, a task that is inherently parallel. Therefore, those applications are already written in a multithreaded style to interface sensors and actuators and execute control low code at different periods. These multithreaded applications are good candidates for CMP systems. With future many-core systems it will even be possible that each thread has its own core to execute. In that case, CPU scheduling, with the scheduling overhead and the predictability issues due to task preemption, disappears and only the memory accesses need to be (statically) scheduled.

II. Related Work

The basis of a time-predictable CMP system is a time-predictable processor. In the following section several approaches to processors designed for real-time systems are presented. Time-predictable chip-multiprocessing is a very recent research topic. Therefore, there are not yet so many publications available.

Edwards and Lee argue: “It is time for a new era of processors whose temporal behavior is as easily controlled as their logical function" [5]. A first simulation of their precision timed (PRET) architecture is presented in [10]. PRET implements a RISC pipeline and performs chip-level multithreading for six threads to eliminate data forwarding and branch prediction. Scratchpad memories are used instead of instruction and data caches. The shared main memory is accessed via a TDMA scheme, called the memory wheel. A resent version of PRET [4] defines time-predictable access to DRAM by assigning each thread a dedicated bank in the memory chips. The access to the individual banks is

pipelined and the access time fixed. As the memory banks are not shared between threads, thread communication has to be performed via the shared scratchpad memory. Although the PRET architecture is not a CMP system, the concepts used in the multi-threaded pipeline can also be applied to a CMP system. We intend to evaluate the PRET memory controller with our CMP system. Each bank will be assigned to a set of CPUs. Within this set we will perform a TDMA based memory arbitration.

Our design can be seen as an instance of the PRET architecture [11]. However, we leave the name PRET to the original Berkeley design. The main difference between our proposal and PRET is that we focus on time predictability and PRET on repeatable timing. In our opinion a time-predictable architecture does not need to provide repeatable timing as long as WCET analysis can deliver tight WCET bounds.

Heckmann et al. provide examples of problematic processor features in [7]. The most problematic features found are the replacement strategies for set-associative caches. A pseudo-round-robin replacement strategy effectively renders the associativity useless for WCET analysis. In conclusion Heckmann et al. suggest the following restrictions for time-predictable processors: (1) separate data and instruction caches; (2) locally deterministic update strategies for caches; (3) static branch prediction; and (4) limited out-of-order execution. The authors argue for restriction of processor features of actual processors (of the time) for embedded systems. In contrast to that proposal, we also provide additional features, such as the split cache, for a time-predictable processor.

Whitham argues that the execution time of a basic block has to be independent of the execution history [29]. As a consequence his MCGREP architecture reduces pipelining to two stages (fetch and execute) and avoids caches all together. To reduce the WCET, Whitham proposes to implement the time critical functions in microcode on a reconfigurable function unit (RFU). With several RFUs, it is possible to explicitly extract instruction level parallelism (ILP) from the original RISC code. Whitham and Audsley extend the MCGREP architecture with a trace scratchpad [30]. The trace scratchpad caches microcode and is placed after the decode stage. The trace scratchpad has to be explicitly started and the scratchpad has to be loaded under program control. The authors extract ILP at the microcode level and schedule the instructions statically – similar to a very long instruction word (VLIW) architecture.

Superscalar out-of-order processors can achieve higher performance than in-order designs, but due to the dynamic allocation of processor resources it is difficult to predict the WCET. Whitham and Audsley present modifications to out-of-order processors to achieve time-predictable operation [32]. Virtual traces allow static WCET analysis, which is performed before execution. Those virtual traces are formed within the program and constrain the out-of order scheduler built into the CPU to execute deterministically.

In comparison to caches, scratchpads are more energy efficient and the access latencies are independent of the preceding memory access pattern. The latter property makes memory access time-predictable which is relevant for hard real-time systems and WCET analysis. [31] introduces the scratchpad memory management unit (SMMU) as an enhancement to scratchpad technology. This proposed solution does not require whole-program pointer analysis and makes load and store operation time-predictable.

The scope of the Connective Autonomic Real-time System-on-Chip (CAR-SoC) [15] project is to integrate the CarCore in a SoC that supports autonomic computing principles for the use in real-time capable autonomic systems. The multithreaded embedded processor executes the time critical application concurrently to a number of helper threads that monitor the system and provide autonomic or organic computing features such as self-configuration, self-healing, self-optimization and self-protection.

Multi-Core Execution of Hard Real-Time Applications Supporting Analysability (MERASA) is a European Union project that aims for multicore processor designs in hard real-time embedded systems. How a single threaded in-order superscalar processor can be enhanced to provide hardware multithreading is described in [13]. By strictly prioritizing multithreading capabilities and completely isolating threads it is possible to reach a deterministic behavior for tight WCET analysis. The CarCore, a multithreaded embedded processor, supports one hard real-time thread to be executed while several non real-time threads run concurrently in the background. The MERASA processor is a CMP system where a simplified version of the CarCore processor is used. The shared memory is accessed via a shared bus.

Not that many papers are available on the design of time-predictable CMP systems. A recent paper discusses some high-level design guidelines [3]. To simplify WCET analysis (or even make it feasible) the architecture shall be *timing compositional*. That means that the architecture has no timing anomalies or unbounded timing effects [12]. The Java processor JOP, used in the proposed time-predictable CMP, fulfills those properties. For CMP systems the authors of [3] argue for bounded access delays on shared resources. This is in our opinion best fulfilled by a TDMA based arbitration scheme.

III. Time-predictable Processors

The basic building block for a time-predictable CMP is a time-predictable processor. In the following we present two architectures that can be used: the Java processor JOP, which is already in industrial us; and the VLIW processor Patmos, which is currently in development.

A. The Java Processo JOP

The Java processor JOP [19] is a hardware implementation of the Java virtual machine (JVM). The JVM bytecodes are the native instruction set of JOP. The main advantage of directly executing bytecode instructions is that WCET analysis can be performed at the bytecode level. The WCET tool WCA [23] is part of the JOP distribution. The Java classfile, which is the Java executable format, contains all symbolic information

of the original program.[1] Having this symbolic information available is of a great help for a WCET analysis tool.

The pipeline and the microcode of JOP has been designed to avoid timing dependencies between bytecode instructions. The execution time of the bytecodes are known cycle accurate. If a bytecode instruction accesses main memory (e.g., object field load), the execution time depends on the memory latency and in the case of a CMP system on the memory arbitration. The timing influence of a TDAM based memory arbiter has been integrated into the timing model of the individual bytecodes for the WCET analysis tool [14].

JOP uses split load instructions to partially hide the memory latency. A split load instruction consists of two instructions: one that triggers the actual memory load and a second one that reads the result. Between those two instructions other, non memory accessing, instructions can be executed. In the microcode of JOP this feature is used to hide some of the memory latency.

JOP contains a method cache [16] for the bytecode instructions, a stack cache [17] for the stack allocated data, and a split-cache for heap allocated data [20], [22]. A special object cache has been integrated into the WCET analysis tool [9] and is currently under development.

JOP is open-source under the GNU GPL and available from http://www.jopdesign.com/. The source distribution also contains the WCET analysis tool WCA. The uniprocessor version of JOP has been in use in industrial applications. The CMP version of JOP was used as the platform for architecture research within the EC funded project JEOPARD [25] on Java for parallel real-time development.

B. The VLIW Processor Patmos

Processors for future embedded systems need to be time-predictable *and* provide a reasonable worst-case performance. Therefore, we develop a VLIW pipeline with special designed caches to provide good single thread performance. To enhance multi-threaded worst-case performance the VLIW pipeline is duplicated in a CMP design.

The time-predictable processor Patmos is one approach to attack the complexity issue of WCET analysis. Patmos is a static scheduled, dual-issue RISC processor that is optimized for real-time systems. All instruction delays are visible at the instruction set architecture. This decision puts more burden on the compiler, but simplifies the WCET analysis tool.

Access to main memory is done via a split load, where one instruction starts the memory read and another instruction explicitly waits for the result. Although this increases the number of instructions to be executed, instruction scheduling can use the split accesses to hide memory access latencies deterministically.

A major challenge for the WCET analysis is the memory hierarchy with multiple levels of caches. We attack this issue by caches that are especially designed for WCET analysis. For instructions we adopt the method cache from JOP [16], which operates on whole functions/methods and thus simplifies the modeling for WCET analysis. This cache organization simplifies the pipeline and the WCET analysis as instruction cache misses can only happen at call or return instructions.

Furthermore, we propose a split-cache architecture [20] for data, offering dedicated caches for the stack area, constants, static data, heap allocated objects, as well as a compiler managed scratchpad memory. Data allocated on the stack is served by a direct mapped stack cache, heap allocated data in a highly associative data cache, and constants and static data in a set associative cache. Only the cache for heap allocated data and static data needs a cache coherence protocol for a CMP configuration of Patmos. Each of these mechanism specifically allows predictable caching of its related data, while at the same time supporting WCET analysis.

Furthermore, a scratchpad memory can also be used to store frequently accessed data. To distinguish between the different caches, Patmos implements typed load and store instructions. The type information is assigned by the compiler (e.g., the compiler already organizes the stack allocated data).

IV. Shared Memory Arbitration

The individual cores of a CMP system share the access bandwidth to the main memory. To build a time-predictable CMP, we need to schedule the access to the main memory in a time-predictable way. A predictable scheduling can only be time based, where each core receives a fixed time slice. This scheduling scheme is called time-division multiple access (TDMA). The execution time of un-cached loads and stores and the cache miss penalty depend on this schedule and therefore, for accurate WCET analysis, the complete schedule needs to be known. A TDMA schedule with equal sized slots has been integrated into the WCET analysis tool for JOP. As far as we know the JOP CMP system [14] is the first time-predictable CMP that includes a WCET analysis tool.

The individual time slices need not be equally sized. Especially on a many-core system the TDMA schedule can be optimized for the application. Assuming that enough cores are available, we propose a CMP model with a single thread per core. In that case, thread switching with the scheduling overhead and the hard to predict cache trashing though preemption disappears. Furthermore, no schedulability analysis has to be performed.

Since each processor executes only a single thread, the WCET of that thread can be as long as its deadline. When the period of a thread is equal to its deadline, 100% utilization of individual cores are feasible, which is in general not possible with priority based scheduling.

For threads that have enough slack time left, we can increase the WCET by decreasing their share of the bandwidth on the memory bus. Other threads with tighter deadlines can, in turn, use additional memory bandwidth in order to reduce their WCET. The TDMA schedule is the input for the WCET analysis and the WCET analysis verifies if all tasks will

[1]There exists even decompilers, which generate reasonable readable Java source from the classfiles.

meet their deadline. Finding a static TDMA schedule for an application is thus an iterative optimization problem [24].

V. Caching

Several cores competing for access to the same memory increase the pressure to reduce accesses to the shared main memory. In standard computer architecture several levels of caches, core local first level caches and shared caches for the other levels, are used. However, for a time-predictable architecture the cache need to be analyzable. A cache shared between different threads and serving instructions and data requests is practically not analyzable. Therefore, only core local caches provide the timing independency between the threads running on different cores.

On a CMP system some data needs to be hold cache coherent. However, not all data is shared between threads. Stack allocated data, constants, and class type information (e.g., the method dispatch table) usually needs no cache coherence. Therefore, splitting the cache for different data areas simplifies some of the caches.

Furthermore, a single data cache for different data areas can be the reason for very conservative WCET bounds. A single access with an unknown address destroys all abstract cache information on one way of a set-associative cache. In the case of a direct mapped cache the abstract cache state for all the whole cache is unknown. Heap allocated objects and arrays are a prime candidate for this issue. Their address is usually only known at runtime and not at static program analysis time. We argue that accesses to data with statically unknown addresses shall not be cached at all or go to a dedicated cache. If this object cache is highly associative, some accesses can be tracked symbolically in the WCET analysis [9].

For a time-predictable CMP design we suggest to use following cache architecture:

- A method cache for the instructions that caches full methods/functions (non cache coherent).
- A direct mapped stack cache for stack allocated data (non cache coherent).
- A cache for constants and class information (direct mapped or medium associativity, not cache coherent).
- A cache for probably shared static data (medium associativity, cache coherent).
- A highly associative cache for heap allocated objects (cache coherent).
- A few prefetch buffers for arrays, which explore primary spatial locality (cache coherent).

The delegation of load and store instructions to the dedicated caches is simple in a Java processor, as the bytecode contains enough information about the memory area. For a RISC style load/store architecture either typed load/store instructions can be introduced or the MMU can be programmed to map the different data areas to address ranges that decide on the cache type.

A common approach to avoid data caches is to use an on-chip memory called scratchpad memory, where data allocation is under program control. This program managed memory entails a more complicated programming model, although it can be automatically partitioned [1], [26]. Exposing the scratchpad memory at the language level can further help to optimize the time-critical path of the application. The real-time specification for Java (RTSJ) [2] has the notation of scoped memory, which serves as temporal allocation context to avoid garbage collection. This scoped memory is an ideal candidate to represent core local scratchpad memories [27].

VI. Alternative Communication Paths

With the increase in number of cores, the memory bandwidth and the cache-coherence protocol are becoming limiting factors for the scalability of CMP systems. We have to reconsider other forms of on-chip communication for further performance scaling. In the 80's the architecture of Transputers [28] provided hardware support for the programming model of Communicating Sequential Processes (CSP [8]). CSP are processes with a clear definition of communication based on message passing. The communication primitives also serve as synchronization points.

In [6] we have presented the design of hardware support for CSP based on-chip communication. The individual cores are connected to a ring based network-on-chip (NoC). To make this (shared) NoC predictable, we applied the same design principle to the NoC as for the memory controller: the access to the NoC is TDMA based. In our case, each core has a dedicated clock cycle where it is allowed to insert a single packet into the ring network. The NoC runs at the same clock speed as the individual cores. For a quite small CMP system, consisting just of three cores, the communication between the cores via the TDMA based NoC is up to 11 times faster than communication via the shared memory.

VII. Conclusion

In this paper we have presented an approach to design a time-predictable chip-multiprocessor. The basis of a time-predictable CMP is a time-predictable core. Dynamic features, which use a large execution history to increase the average case performance, are problematic for WCET analysis. Especially interferences between different features result in a state space explosion for the analysis. The presented architectures, the Java processor JOP and the dual-issue VLIW processor Patmos, are in-order pipelines without implicit instruction dependencies. Both architectures use a method cache containing whole methods and a data cache that is split for stack allocated data, constants, and heap allocated data.

A streamlined, time-predictable processor is quite small. Therefore, we can regain performance by the exploration of thread level parallelism in embedded applications with a replication of the processor in a CMP architecture. The key component for a time-predictable CMP is static scheduling of the accesses to the shared memory. The static schedule guarantees that thread execution times on different cores are independent of each other.

For future many-core systems we propose to execute just a single thread per processor core. In that case CPU scheduling

disappears. Only the WCET of the individual threads has to be less than their deadline. With this approach the individual threads can execute at 100% core utilization. With this configuration the memory access can be statically scheduled according to the application needs. With non-uniform time slices, the arbiter schedule can be adapted to balance the utilization of the individual cores.

The Java processor JOP, which is the basis of the presented paper, has been used with success to implement several commercial real-time applications [18]. JOP is open-source under the GNU GPL and all design files (including the CMP version), the documentation, and the WCET analysis tool are available at http://www.jopdesign.com/.

ACKNOWLEDGMENT

The author would like to thank Christof Pitter for the implementation of the CMP version of JOP during his PhD thesis and Wolfgang Puffitsch for the active development on JOP, which includes a first version of the split-cache design.

REFERENCES

[1] Federico Angiolini, Luca Benini, and Alberto Caprara. Polynomial-time algorithm for on-chip scratchpad memory partitioning. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES-03)*, pages 318–326, New York, October 30 November 01 2003. ACM Press.

[2] Greg Bollella, James Gosling, Benjamin Brosgol, Peter Dibble, Steve Furr, and Mark Turnbull. *The Real-Time Specification for Java*. Java Series. Addison-Wesley, June 2000.

[3] Christoph Cullmann, Christian Ferdinand, Gernot Gebhard, Daniel Grund, Claire Maiza, Jan Reineke, Benoît Triquet, and Reinhard Wilhelm. Predictability considerations in the design of multi-core embedded systems. In *Proceedings of Embedded Real Time Software and Systems*, May 2010.

[4] Stephen A. Edwards, Sungjun Kim, Edward A. Lee, Isaac Liu, Hiren D. Patel, and Martin Schoeberl. A disruptive computer design idea: Architectures with repeatable timing. In *Proceedings of IEEE International Conference on Computer Design (ICCD 2009)*, Lake Tahoe, CA, October 2009. IEEE.

[5] Stephen A. Edwards and Edward A. Lee. The case for the precision timed (PRET) machine. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 264–265, New York, NY, USA, 2007. ACM.

[6] Flavius Gruian and Martin Schoeberl. NoC-based CSP support for a Java chip multiprocessor. In *Proceedings of the 28th Norchip Conference*, Tampere, Finland, November 2010. IEEE Computer Society.

[7] Reinhold Heckmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The influence of processor architecture on the design and results of WCET tools. *Proceedings of the IEEE*, 91(7):1038–1054, Jul. 2003.

[8] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[9] Benedikt Huber, Wolfgang Puffitsch, and Martin Schoeberl. WCET driven design space exploration of an object caches. In *Proceedings of the 8th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2010)*, pages 26–35, New York, NY, USA, 2010. ACM.

[10] Ben Lickly, Isaac Liu, Sungjun Kim, Hiren D. Patel, Stephen A. Edwards, and Edward A. Lee. Predictable programming on a precision timed architecture. In Erik R. Altman, editor, *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008)*, pages 137–146, Atlanta, GA, USA, October 2008. ACM.

[11] Isaac Liu, Jan Reineke, and Edward A. Lee. A PRET architecture supporting concurrent programs with composable timing properties. In *Signals, Systems and Computers, 2010 Conference Record of the Forty-Four Asilomar Conference on*, Novemeber 2010.

[12] Thomas Lundqvist and Per Stenström. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS 1999)*, pages 12–21, Washington, DC, USA, 1999. IEEE Computer Society.

[13] Jörg Mische, Irakli Guliashvili, Sascha Uhrig, and Theo Ungerer. How to enhance a superscalar processor to provide hard real-time capable in-order smt. In *23rd International Conference on Architecture of Computing Systems (ARCS 2010)*, pages 2–14, University of Augsburg, Germany, February 2010. Springer.

[14] Christof Pitter and Martin Schoeberl. A real-time Java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst.*, 10(1):9:1–34, 2010.

[15] Theo Ungerer Sascha Uhrig, Stefan Maier. Toward a processor core for real-time capable autonomic systems. In *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology*, pages 1–4, University of Augsburg, Germany, 2005.

[16] Martin Schoeberl. A time predictable instruction cache for a Java processor. In *On the Move to Meaningful Internet Systems 2004: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2004)*, volume 3292 of *LNCS*, pages 371–382, Agia Napa, Cyprus, October 2004. Springer.

[17] Martin Schoeberl. Design and implementation of an efficient stack machine. In *Proceedings of the 12th IEEE Reconfigurable Architecture Workshop (RAW2005)*, Denver, Colorado, USA, April 2005. IEEE.

[18] Martin Schoeberl. Application experiences with a real-time Java processor. In *Proceedings of the 17th IFAC World Congress*, pages 9320–9325, Seoul, Korea, July 2008.

[19] Martin Schoeberl. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, 54/1–2:265–286, 2008.

[20] Martin Schoeberl. Time-predictable cache organization. In *Proceedings of the First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009)*, pages 11–16, Tokyo, Japan, March 2009. IEEE Computer Society.

[21] Martin Schoeberl. Time-predictable computer architecture. *EURASIP Journal on Embedded Systems*, vol. 2009, Article ID 758480:17 pages, 2009.

[22] Martin Schoeberl, Wolfgang Puffitsch, and Benedikt Huber. Towards time-predictable data caches for chip-multiprocessors. In *Proceedings of the Seventh IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2009)*, number LNCS 5860, pages 180–191. Springer, November 2009.

[23] Martin Schoeberl, Wolfgang Puffitsch, Rasmus Ulslev Pedersen, and Benedikt Huber. Worst-case execution time analysis for a Java processor. *Software: Practice and Experience*, 40/6:507–542, 2010.

[24] Martin Schoeberl and Peter Puschner. Is chip-multiprocessing the end of real-time scheduling? In *Proceedings of the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, Dublin, Ireland, July 2009. OCG.

[25] Fridtjof Siebert. JEOPARD: Java environment for parallel real-time development. In *Proceedings of the 6th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2008)*, pages 87–93, New York, NY, USA, 2008. ACM.

[26] Manish Verma and Peter Marwedel. Overlay techniques for scratchpad memories in low power embedded processors. *IEEE Trans. VLSI Syst*, 14(8):802–815, 2006.

[27] Andy Wellings and Martin Schoeberl. Thread-local scope caching for real-time Java. In *Proceedings of the 12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2009)*, pages 275–282, Tokyo, Japan, March 2009. IEEE Computer Society.

[28] Colin Whitby-Strevens. The transputer. *SIGARCH Comput. Archit. News*, 13(3):292–300, 1985.

[29] Jack Whitham. *Real-time Processor Architectures for Worst Case Execution Time Reduction*. PhD thesis, University of York, 2008.

[30] Jack Whitham and Neil Audsley. Using trace scratchpads to reduce execution times in predictable real-time architectures. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*, pages 305–316, April 2008.

[31] Jack Whitham and Neil Audsley. Implementing time-predictable load and store operations. In *Proceedings of the International Conference on Embedded Software (EMSOFT 2009)*, 2009.

[32] Jack Whitham and Neil Audsley. Time-predictable out-of-order execution for hard real-time systems. *IEEE Transactions on Computers*, 59(9):1210–1223, 2010.